Paleti Krishnasai
CED18I039

OS MID SEM NOV 1

Question :
Dear PALETI KRISHNASAI,
Question: Develop a Multiprocessing Version of Radix Sort algorithm and compare it with the performance (execution time) of bubble and insertion sort algorithms. The demonstration should display the passes of the respective sorting strategies. The comparison is against the Multiprocessing Radix sort with sequential versions of Bubble and Insertion Sort Algorithm. Ensure that the testing explores large sized arrays, random distribution of elements. As an addon it would be preferred to have a data creator code which initializes an array of user required size randomly given some boundary conditions.

Answer :

Radix Sort (Multiprocessing)

Radix Sort sorts the elements by initially grouping the individual digits of the same place value. The idea of Radix Sort is to do digit by digit sort *starting from least significant digit(LSD) to the most significant digit(MSD)*, according to their increasing/decreasing order.

Algorithm

```
Radix_sort (list, n)
shift = 1
for loop = 1 to keysize do
   for entry = 1 to n do
   bucketnumber = (list[entry].key / shift) mod 10
   append (bucket[bucketnumber], list[entry])
list = combinebuckets()
shift = shift * 10
```

Paleti Krishnasai
CED18I039

In a multiprocessing setup , the total number of passes have been split up , where half the passes are done in the child process and the trailing half are done in the parent process.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#define MAX 100000

int get_max(int a[], int n)
{
   int max = a[0];
   for (int i=1;i<n;i++)
       if(a[i]>max)
           max = a[i];
   return max;
}

int main()
{
   srand(time(0));
   struct timeval start, stop;
   double secs = 0;

   int output_fds = open("./output.txt",O_WRONLY | O_APPEND);
   dup2(output_fds,STDOUT_FILENO);
   int size,array[MAX];
   printf("Enter the size of the list : \n");
   scanf("%d",&size);
   for(int i=0;i<size;i++)
```

```
  {
      array[i] = rand()%(size+1);
  }
/* for (i = 0; i < 100; i++)
  {
      printf("%d --->  %d\n",i+1,array[i]);
  }
  */
  gettimeofday(&start, NULL);
  int bucket[size][size], bucket_count[size];
  int i, j, k, r, NOP = 0, divisor = 1, lar, pass;
  lar = get_max(array, size);

  int pid = vfork();
  if(pid==0)
  {

      while(lar > 0)
      {
          NOP++;
          lar /= 10;

      }
      for(pass=0;pass<NOP/2;pass++)
      {
          for(i=0;i<size;i++)
          {
              bucket_count[i] = 0;
          }
          for(i =0;i<size;i++)
          {
              r = (array[i]/divisor)%10;
              bucket[r][bucket_count[r]] = array[i];
              bucket_count[r] += 1;

          }
          i=0;
          for(k=0;k<size;k++)
          {
              for (j=0;j<bucket_count[k];j++)
```

```
            {
                array[i] = bucket[k][j];
                i++;
            }
        }
        divisor *= 10;
        printf("PASS %d : \n",pass+1);
        for(i =0;i<size;i++)
        {
            printf("%d ",array[i]);
        }
        printf("\n");
    }
    exit(0);
}
if(pid>0)
{
    wait(NULL);
    for (pass = NOP/2; pass < NOP; pass++)
    {
        for (i = 0; i < size; i++)
        {
            bucket_count[i] = 0;
        }
        for (i = 0; i < size; i++)
        {
            r = (array[i] / divisor) % 10;
            bucket[r][bucket_count[r]] = array[i];
            bucket_count[r] += 1;
        }
        i = 0;
        for (k = 0; k < size; k++)
        {
            for (j = 0; j < bucket_count[k]; j++)
            {
                array[i] = bucket[k][j];
                i++;
            }
        }
        divisor *= 10;
```

```
            printf("\n");
            printf("PASS %d : \n", pass + 1);
            printf("\n");
            for (i = 0; i < size; i++)
            {
                printf("%d ", array[i]);
            }
            printf("\n");
        }
    }

    gettimeofday(&stop, NULL);
    secs = (double)(stop.tv_usec - start.tv_usec) / 1000000 +
(double)(stop.tv_sec - start.tv_sec);

    printf("\n");
    printf("Sorted list : \n");
    printf("\n");
    for (i=0;i<size;i++)
    printf("%d ",array[i]);
    printf("\n");
    printf("Execution time:  %f\n",secs);

    return 0;
}
```

Paleti Krishnasai
CED18I039

Insertion and Bubble Sort :

Classic Serial implementation of bubble and insertion sort .

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#define MAX 100000

void insertionSort(int arr[], int n)
{
   int i, key, j;
   for (i = 1; i < n; i++)
   {
       key = arr[i];
       j = i - 1;

       /* Move elements of arr[0..i-1], that are
         greater than key, to one position ahead
         of their current position */
       while (j >= 0 && arr[j] > key)
       {
           arr[j + 1] = arr[j];
           j = j - 1;
       }
       arr[j + 1] = key;
       for (int k = 0; k < n; ++k)
           printf(" %d", arr[k]);
       printf("\n");
   }
}
```

Paleti Krishnasai
CED18I039

```c
void printArray1(int arr[], int n)
{
   int i;
   for (i = 0; i < n; i++)
       printf("%d ", arr[i]);
   printf("\n");
}
void swap(int *xp, int *yp)
{
   int temp = *xp;
   *xp = *yp;
   *yp = temp;
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
   int i, j, pass = 0;
   for (i = 0; i < n - 1; i++)
   {
       // Last i elements are already in place
       for (j = 0; j < n - i - 1; j++)
           if (arr[j] > arr[j + 1])
               swap(&arr[j], &arr[j + 1]);
       //pass++;
       //printf("  After pass %d:", pass);
       for (int k = 0; k < n; ++k)
           printf(" %d", arr[k]);
       printf("\n");
   }
}

/* Function to print an array */
void printArray2(int arr[], int size)
{
   int i;
   for (i = 0; i < size; i++)
       printf("%d ", arr[i]);
   printf("\n");
}
```

Paleti Krishnasai
CED18I039

```c
int main()
{
   srand(time(0));
   struct timeval start, stop;
   double secs = 0;
   //int output_fds = open("./output.txt", O_WRONLY);
   //dup2(output_fds, STDOUT_FILENO);
   int size, array1[MAX], array2[MAX];
   printf("Enter the size of the list : \n");
   scanf("%d", &size);
   for (int i = 0; i < size; i++)
   {
       array1[i] = rand() % size+1;
       array2[i] = array1[i];
   }
   printf("\nArray:");
   for (int i = 0; i < size; i++)
   {
       printf(" %d", array1[i]);
   }
   printf("\n");
   printf("----------------------------------------\n\n");

   gettimeofday(&start, NULL);

   insertionSort(array1, size);
   printf("\n");
   printArray1(array1, size);
   printf("----------------------------------------\n\n");
   printf("\n");
   bubbleSort(array2, size);
   printf("\n");
   printArray2(array2, size);
   gettimeofday(&stop, NULL);
   secs = (double)(stop.tv_usec - start.tv_usec) / 1000000 +
(double)(stop.tv_sec - start.tv_sec);

   printf("\n");
   printf("Execution time:  %f\n", secs);
```

```
    return 0;
}
```

Output :
Will print in a file.

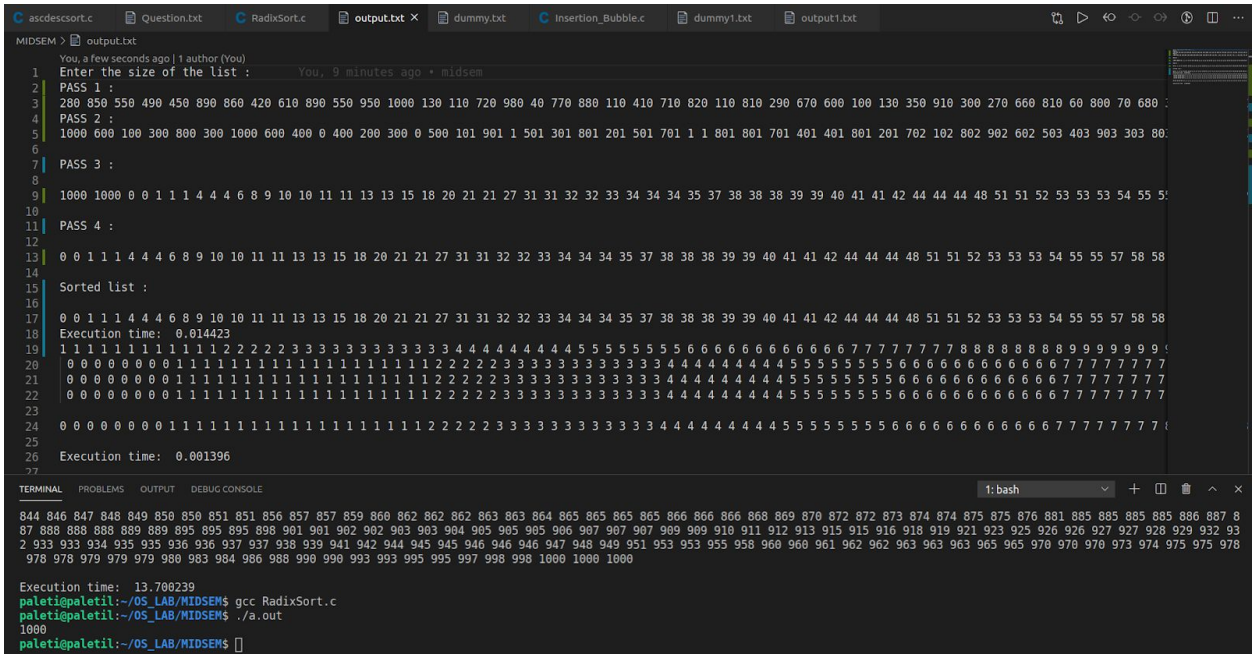
dummy.txt = terminal output of radix sort copied to a file.
dummy1.txt = terminal output of bubble and insertion sort copied to a file.
output.txt = output to a file using dup2() system call.
output1.txt = output of bubble and insertion sort to a file using dup2() system call. // need to comment out the respective lines in the program to get this functionality.

Execution times have been compared and it can be noted that the multiprocessing radix sort is slightly slower than the serial implementation of bubble and insertion sort for input sizes of 100 or lesser.

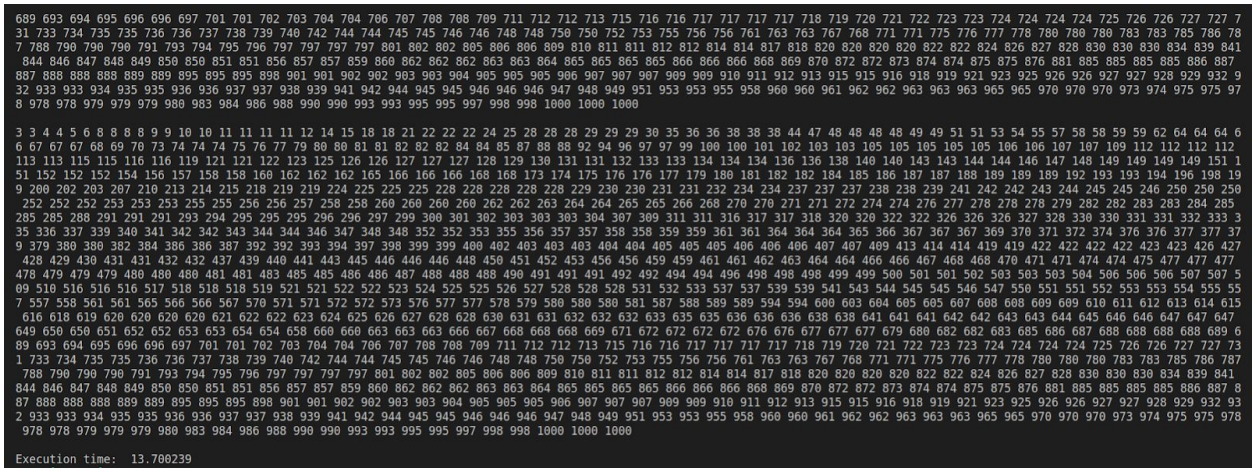
But when input size goes to 1000 , then we can see significant difference in execution time .(pic shown below)

FEW screenshots are shown below.(codes and required files will be attached in a zip file)

Paleti Krishnasai
CED18I039



Radix sort : Execution time: 0.001396



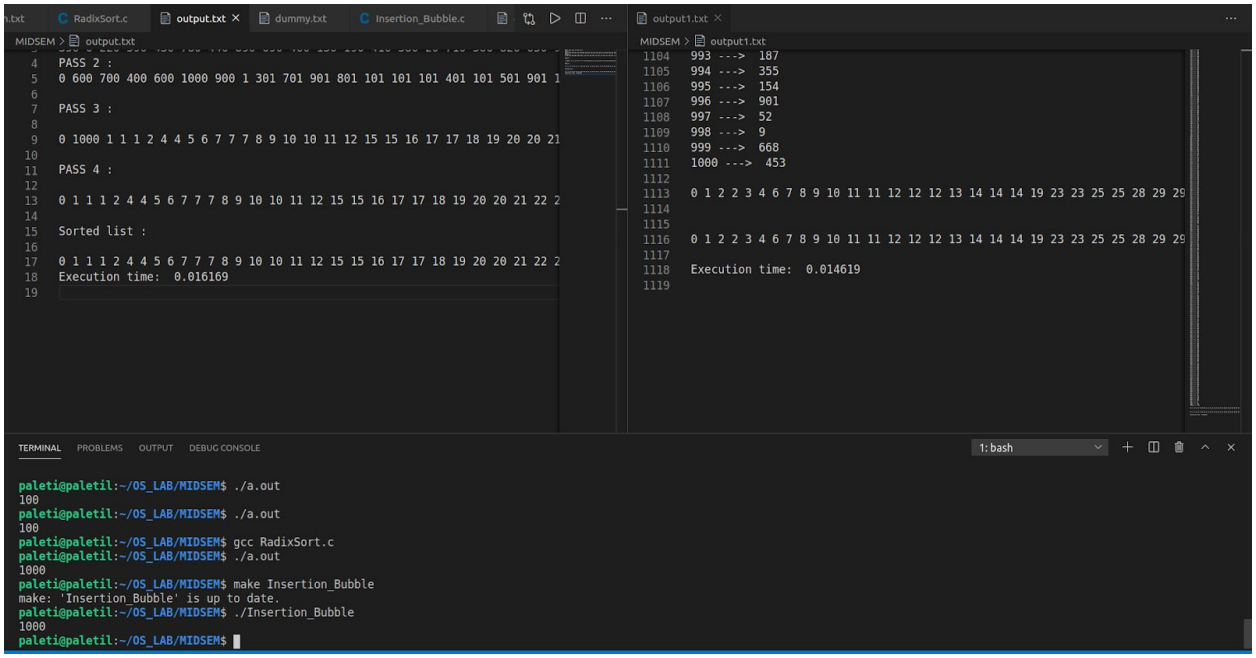
Insertion and bubble sort : Execution time: 13.700239

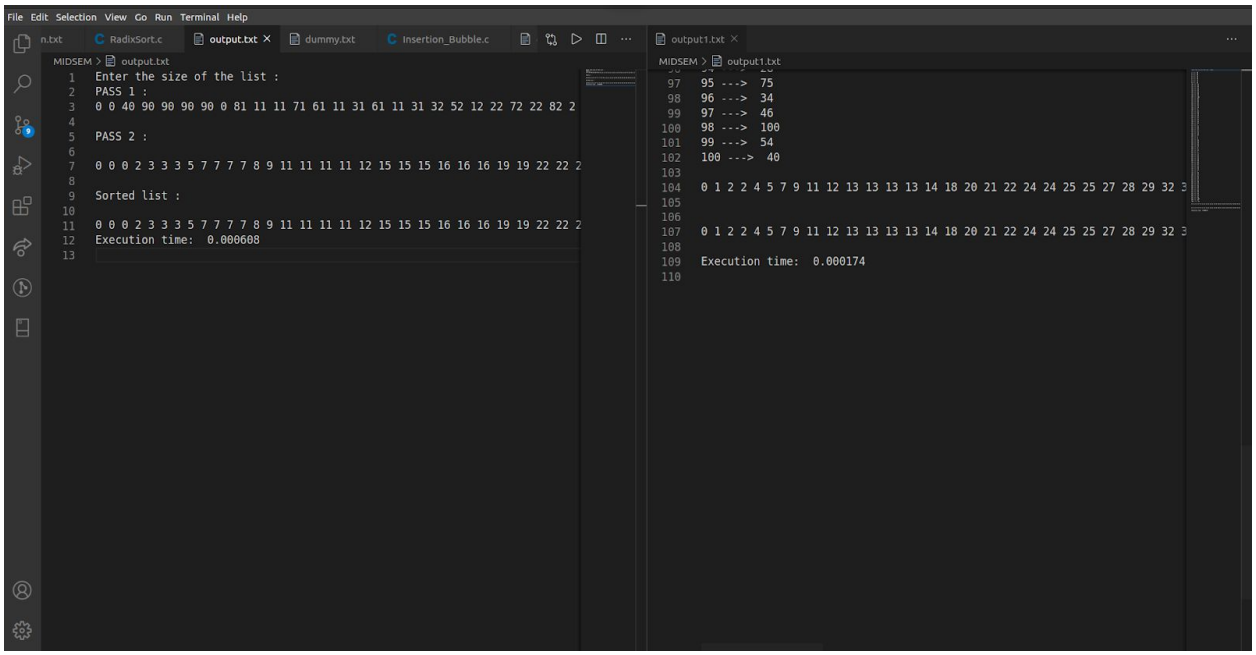
Paleti Krishnasai
CED18I039

Printing of passes for a small input size for readability (insertion and bubble sort)

```
paleti@paletil:~/OS_LAB/MIDSEM$ ./Insertion_Bubble
Enter the size of the list :
10

Array: 10 10 4 3 10 5 10 3 8 10
---------------------------------------

 10 10 4 3 10 5 10 3 8 10
 4 10 10 3 10 5 10 3 8 10
 3 4 10 10 10 5 10 3 8 10
 3 4 10 10 10 5 10 3 8 10
 3 4 5 10 10 10 10 3 8 10
 3 4 5 10 10 10 10 3 8 10
 3 3 4 5 10 10 10 10 8 10
 3 3 4 5 8 10 10 10 10 10
 3 3 4 5 8 10 10 10 10 10

3 3 4 5 8 10 10 10 10 10
---------------------------------------


 10 4 3 10 5 10 3 8 10 10
 4 3 10 5 10 3 8 10 10 10
 3 4 5 10 3 8 10 10 10 10
 3 4 5 3 8 10 10 10 10 10
 3 4 3 5 8 10 10 10 10 10
 3 3 4 5 8 10 10 10 10 10
 3 3 4 5 8 10 10 10 10 10
 3 3 4 5 8 10 10 10 10 10
 3 3 4 5 8 10 10 10 10 10

3 3 4 5 8 10 10 10 10 10

Execution time:  0.000175
```

Paleti Krishnasai
CED18I039





Paleti Krishnasai
CED18I039

```
paleti@paletil:~/OS_LAB/MIDSEM$ gcc RadixSort.c
paleti@paletil:~/OS_LAB/MIDSEM$ ./a.out
Enter the size of the list :
100
PASS 1 :
100 30 80 10 50 0 40 10 50 10 30 50 91 21 51 81 51 1 71 51 72 22 52 52 92 12 62 92 92 92 42 2 63 73 63 63 63 93 43 13 94 74 64 14 44 74 14 74 44 54 74 75 45 15 5 55 35 36 66 96 9
6 26 6 6 26 16 86 97 27 47 27 77 7 87 77 47 47 87 47 7 88 28 38 48 68 48 58 98 98 28 99 99 19 19 59 39 69 99 19

PASS 2 :

100 0 1 2 5 6 6 7 7 10 10 10 12 13 14 14 15 16 19 19 19 21 22 26 26 27 27 28 28 30 30 35 36 38 39 40 42 43 44 44 45 47 47 47 47 48 48 50 50 50 51 51 51 52 52 54 55 58 59 62 63 6
 63 64 66 66 68 69 71 72 73 74 74 74 74 75 77 77 80 81 86 87 87 88 91 92 92 92 92 93 94 96 96 97 98 98 99 99 99

PASS 3 :

0 1 2 5 6 6 7 7 10 10 10 12 13 14 14 15 16 19 19 19 21 22 26 26 27 27 28 28 30 30 35 36 38 39 40 42 43 44 44 45 47 47 47 47 48 48 50 50 50 51 51 51 52 52 54 55 58 59 62 63 63 63
64 66 66 68 69 71 72 73 74 74 74 74 75 77 77 80 81 86 87 87 88 91 92 92 92 92 93 94 96 96 97 98 98 99 99 99 100

Sorted list :

0 1 2 5 6 6 7 7 10 10 10 12 13 14 14 15 16 19 19 19 21 22 26 26 27 27 28 28 30 30 35 36 38 39 40 42 43 44 44 45 47 47 47 47 48 48 50 50 50 51 51 51 52 52 54 55 58 59 62 63 63 63
64 66 66 68 69 71 72 73 74 74 74 74 75 77 77 80 81 86 87 87 88 91 92 92 92 92 93 94 96 96 97 98 98 99 99 99 100
Execution time:  0.002326
```